**ARITHMETIC LOGIC UNIT (ALU) DESIGN DOCUMENT**

# Introduction

The contents of this report provide the architecture, description, and future of an Arithmetic Logic Unit (ALU) designed in Verilog. The ALU is the most basic combinational circuit in digital systems. It performs many arithmetic and logic functions, and it is an essential building block in processors and embedded systems. This document focuses on an ALU that was designed and tested in Verilog.

# Objectives

* To design and implement a parameterized ALU in Verilog which performs various arithmetic and logical operations.
* To develop the ALU using clean, modular, and synthesizable Verilog code while adhering to industry best practice.
* To create a detailed test plan that thoroughly tests all features and edge cases of the ALU.
* To create a self-checking testbench comprised of a driver, monitor, and scoreboard that takes input stimuli defined in a text file and verified outputs from the ALU.
* To conduct code coverage analysis to determine if all functionality paths and operations in the design were sufficiently tested.

# Architecture

**Input Pin Description**

|  |  |
| --- | --- |
| PIN | DESCRIPTION |
| RST | Asynchronous active-high reset, used to set all the outputs to zero. |
| CLK | Provides the timing allowing the circuit to perform synchronous operation. |
| CE | Used to latch inputs. |
| INP\_VALID | Tells the ALU which of the operands (opa and opb) are valid to be used for operations. |
| MODE | Determines whether the ALU is in Arithmetic (1) or Logical (0) mode. |
| CMD | Specifies the exact operation to perform within the selected mode (e.g., ADD, SUB, AND, OR). |
| OPA | Input 1 operand of W bit wide for the processing unit. |
| OPB | Input 2 operand of W bit wide for the processing unit. |
| CIN | Carry Input for processing unit. Used in some specific Addition and Subtraction operations. |

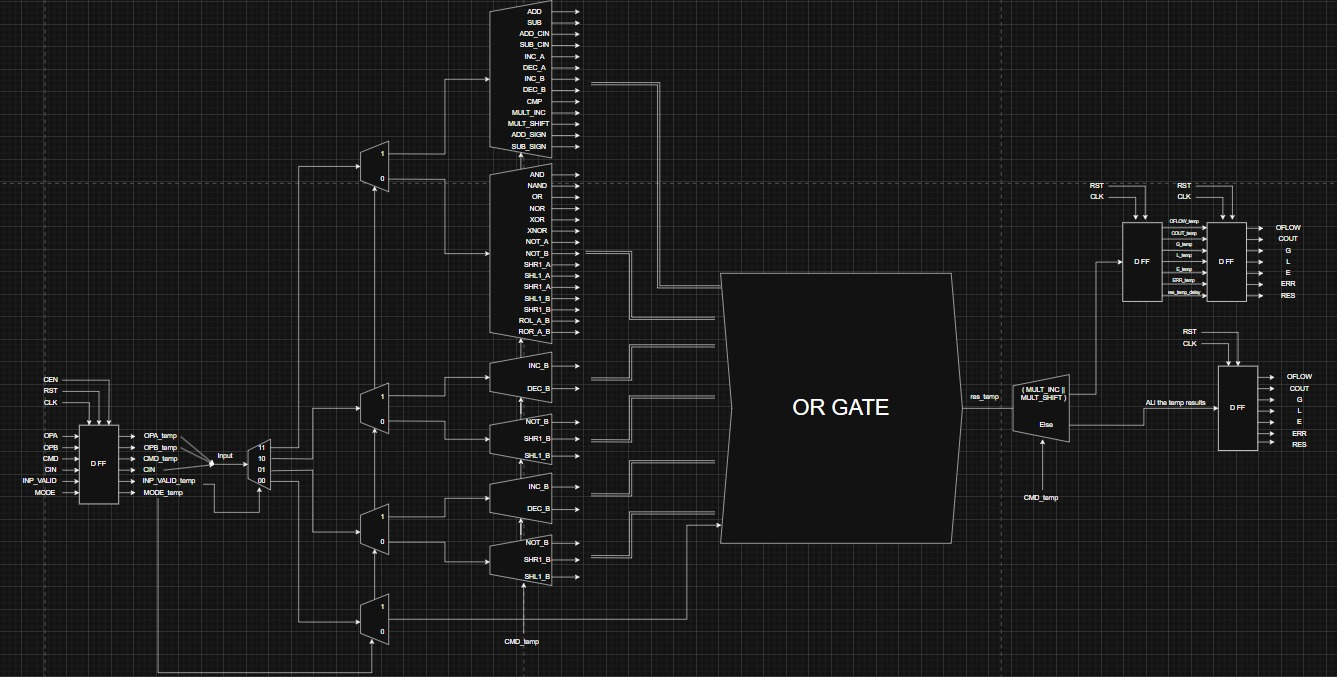
**Output Pin Description**

|  |  |
| --- | --- |
| PIN | DESCRIPTION |
| RES | Result register output of 2W width. |
| OFLOW | This 1-bit signal indicates an output overflow, during Addition/Subtraction. |
| COUT | This is the carry out signal of 1-bit, during Addition / Subtraction. |
| G | This is the comparator output of 1-bit, which indicates that the value of OPA is greater than the value of OPB. |
| L | Determines whether the ALU is in Arithmetic (1) or Logical (0) mode. |
| E | This is the comparator output of 1-bit, which indicates that the value of OPA is equal to the value of OPB. |
| ERR | When CMD is selected as 12 or 13 and mode is logical operation, if 4th ,5th ,6th and 7th bit of OPB are 1, then ERR bit will be 1 else it is high impedance. |

**Design Architecture**

The ALU is designed as a combinational logic block placed between clocked input and output registers for seamless integration in synchronous systems. Inputs are latched on the rising clock edge, processed in a combinational way, and results are stored in output registers, introducing a one-cycle latency.

It supports both arithmetic and logical operations, selected via a mode signal, 4-bit command code and 2-bit INP\_VALID code. The ALU takes up to two operands based on a 2-bit input validity signal, enabling both single-input and dual-input operations depending on its status. Flags like overflow, carry out, comparison results, and error detection are generated alongside outputs. Signed operations, including shifts, rotates, and multiply instructions, are also supported within this ALU module. For multiplication commands a further 1 more delay is introduces meaning a total of 2 clock cycle delay before the output is received.

****

Design Architecture

**Testbench Architecture**

**Packet Architecture**

Packets store the inputs and expected outputs.

1. Testcase Packet (from stimulus.txt)



1. Response Packet

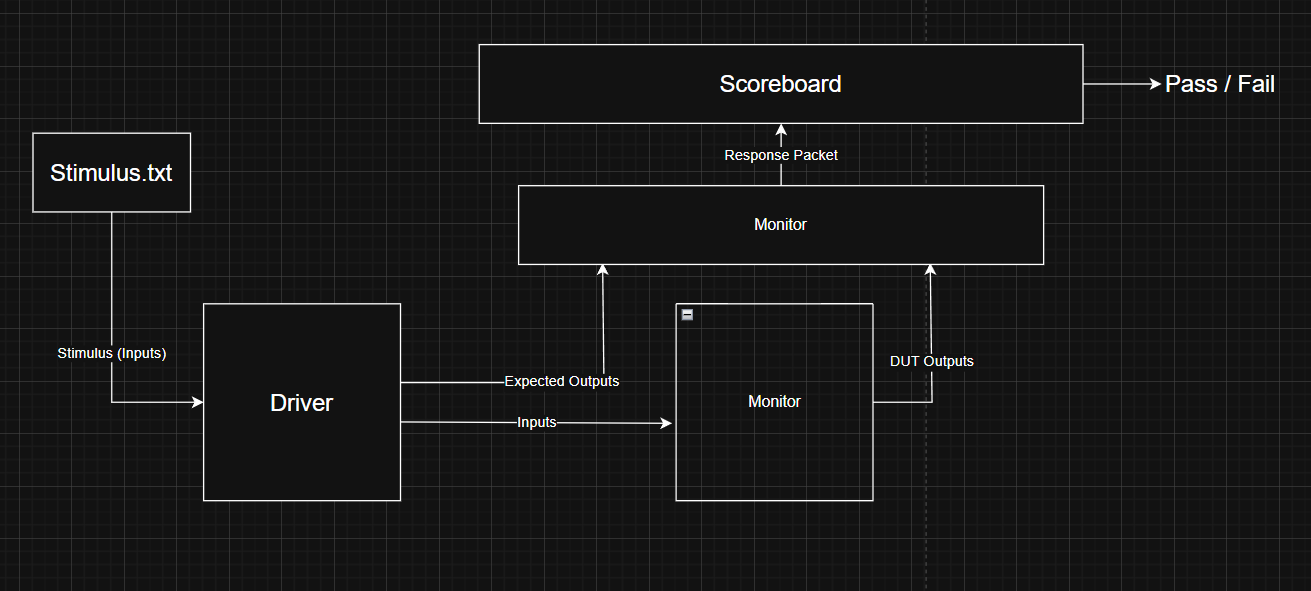
A close up of a box

AI-generated content may be incorrect.

**Testbench Structure**

The testbench is modular and has three main components: Driver, Monitor, and Scoreboard.

1. **Driver**: The driver reads the files containing the stimulus packets (namely stimulus.txt). Each packet will contain input operands, operation codes, and the expected outputs. The driver separates these values and drives the input into the ALU DUT.
2. **Monitor**: The monitor is responsible for observing the inputs into the DUT and recording the actual output that the ALU returns. The monitor records the actual output of the DUT during all stimulus input in a response packet.
3. **Scoreboard**: The scoreboard is responsible for comparing the DUT output (stored in a response packet) from the monitor against the expected outputs from the driver. The scoreboard will report each test case as pass or fail, maintain the number of failed test cases, and issue a final functional coverage report.

****

Testbench Architecture

# Working

The ALU accepts two N-bit operands (opa, opb), a mode signal, and a 4-bit command code. The mode signal determines whether the ALU performs arithmetic (mode = 1) or logical (mode = 0) operations. Based on the selected mode and command, the ALU performs the specified function using internal case structures within an always combinational block.

|  |  |
| --- | --- |
| **Mode** | **Significance** |
| 0 | Logical Mode |
| 1 | Arithmetic Mode |

**Input Handling**

Inputs are validated using a 2-bit inp\_valid signal: -

|  |  |
| --- | --- |
| **INP\_VALID** | **Significance** |
| 00 | Both OPA and OPB are not valid Inputs. |
| 01 | Only OPA is a valid input. |
| 10 | Only OPB is a valid input. |
| 11 | Both OPA and OPB are valid inputs. |

**Operations**

The ALU supports a wide range of Arithmetic and Logical operations: -

|  |  |
| --- | --- |
| Arithmetic Function | Result |
| ADD | OPA + OPB |
| SUB | OPA – OPB |
| ADD\_CIN | OPA + OPB + CIN |
| SUB\_CIN | OPA – OPB – CIN |
| INC\_A | OPA + 1 |
| DEC\_A | OPA – 1 |
| INC\_B | OPB + 1 |
| DEC\_B | OPB – 1 |
| CMP | Compares OPA and OPB and accordingly one of EGL goes high. |
| MULT\_INC | (OPA + 1) \* (OPB + 1) |
| MULT\_SHIFT | (OPA << 1) \* OPB |
| ADD\_SIGN | OPA + OPB (Signed) |
| SUB\_SIGN | OPA – OPB (Signed) |

|  |  |
| --- | --- |
| Logical Function | Result |
| AND | OPA & OPB (bitwise AND) |
| NAND | Bitwise NAND of OPA, OPB |
| OR | Bitwise OR of OPA, OPB |
| NOR | Bitwise NOR of OPA, OPB |
| XOR | Bitwise XOR of OPA, OPB |
| XNOR | Bitwise XNOR of OPA, OPB |
| NOT\_A | NOT of OPA |
| NOT\_B | NOT of OPB |
| SHR1\_A | Shift OPA right by 1 bit |
| SHL1\_A | Shift OPA left by 1 bit |
| SHR1\_B | Shift OPB right by 1 bit. |
| SHL1\_B | Shift OPB left by 1 bit. |
| ROL\_A\_B | Rotate OPA by OPB bits left. If OPB [7:4] any bit is 1 then make err flag high. |
| ROR\_A\_B | Rotate OPA by OPB bits right. If OPB [7:4] any bit is 1 then make err flag high. |

**Status Flags**

To provide detailed operational feedback, the ALU generates several status outputs:

* **Cout**: Carry-out from addition/subtraction.
* **Oflow**: Overflow detection for signed operations.
* **Err**: Invalid input or illegal operation code.
* **G, L, E**: Greater-than, less-than, equal flags for comparisons.

Signed operations use sign bit logic to properly detect and handle overflows. The design also ensures correct behaviour for corner cases like saturated arithmetic and rotate / shift wrap-around.

**Output Logic**

Immediately after the inputs are captured, inside the combinational block the specific command is performed, and the result is stored in a temporary register. On the next clock edge, the result is loaded into result port. However, if it is a multiplication operation its loaded into a second temporary register which is loaded to the output result register on the next clock edge.

# Result

**A screenshot of a computer

AI-generated content may be incorrect.**

Output is received after 1 clock cycle

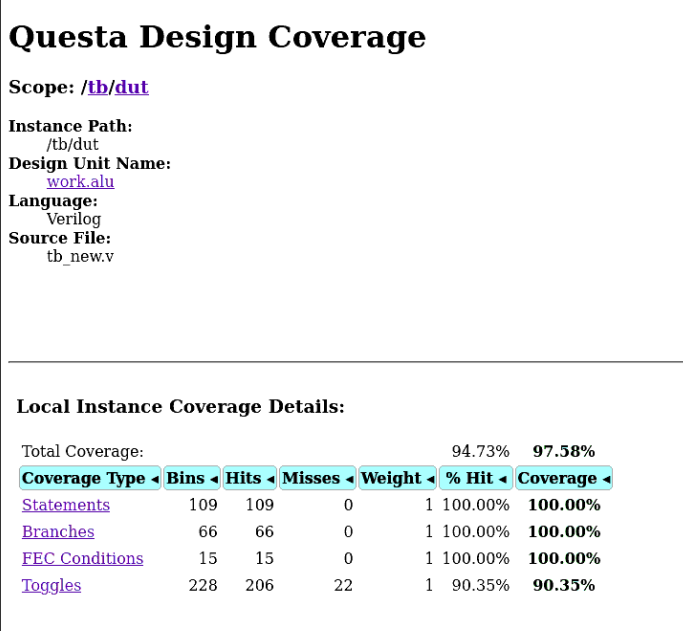
**A green and white lines on a black background

AI-generated content may be incorrect.**

Output received after 2 clock cycles

**Coverage Report**

It shows the percentage of code our testcases have hit and tested.

****

# Conclusion

The Verilog ALU meets all functional requirements with modular, testable, and synthesizable code.

# Future Improvements

* Apply more Design Verification techniques to find and fix even more errors in the design.
* Increase the number of operations.